Analysis of Checkpointing Schemes with Task Duplication

Avi Ziv, *Member*, *IEEE*,
and Jehoshua Bruck, *Senior Member*, *IEEE*

Abstract—This paper suggests a technique for analyzing the performance of checkpointing schemes with task duplication. We show how this technique can be used to derive the average execution time of a task and other important parameters related to the performance of checkpointing schemes. The analysis results are used to study and compare the performance of four existing checkpointing schemes. Our comparison results show that, in general, the number of processors used, not the complexity of the scheme, has the most effect on the scheme performance.

Index Terms—Parallel computing, fault tolerance, checkpointing, task duplication, Markov Reward Model.

1 Introduction

Parallel computing systems provide hardware redundancy that helps to achieve low cost fault tolerance by duplicating the task into more than a single processor and comparing the states of the processors at checkpoints [14]. The usage of checkpoints reduces the time spent in retrying a task in the presence of failures and, hence, reduces the average execution time of a task [3], [18]. Reducing the task execution time is very important in many applications, like real-time systems with hard deadlines and transactions systems, where high availability is required.

In checkpointing schemes, the task is divided into n intervals. At the end of each interval a checkpoint is added, either by the programmer [3] or by the compiler [12]. In the schemes considered here, the checkpoints serve two purposes: detecting faults that occurred during the execution of a task and reducing the time spent in recovering from faults. Fault detection is achieved by duplicating the task into two or more processors and comparing the states of the processors at the checkpoints. We assume that the probability of two faults resulting in identical states is very small, so that two matching states indicate a correct execution. By saving the state of the task at each checkpoint, we avoid the need to restart the task after each fault. Instead, the task can be rolled back to the last correct checkpoint and execution resumed from there, thereby shortening fault recovery. Checkpointing schemes with task duplication can be used to detect and recover from transient faults in multiprocessor systems.

Agrawal [1] describes a fault tolerance scheme, called RAFT (Recursive Algorithm for Fault Tolerance), which achieves fault tolerance by duplicating the computation of a task on two processors. If the results of the two executions do not match, the task is executed again by another processor until a pair of processors produces identical results. The RAFT scheme does not use checkpoints, and every time a fault is detected the task has to be started from its beginning. More recent schemes use checkpointing to avoid reexecution of an entire task [14]. At each checkpoint, the state of the task is stored into a stable memory. If a fault is detected and a rollback is needed, it can be done to the last stored checkpoint, not to the beginning of the task. Different recovery techniques are used by the schemes to shorten the fault recovery time. Examples of such techniques are fault-masking schemes [11], look-back schemes [11], and roll-forward schemes [11], [15]. In [13], Long et al. describe an implementation of a roll-forward scheme on a set of Sun workstations and the experimental results obtained from that system.

Performance analysis is very important when trying to evaluate and compare different schemes or check if a scheme achieves its goals in a certain system. While extensive work has been done on the analysis of checkpointing schemes when checkpoints are used only for fault recovery [4], [6], [7], [10], for checkpointing schemes with task duplication, most authors rely on simulations for performance evaluation [15] or use simplified models [11], [15]. The use of simulation leads to long and time consuming evaluation, and does not allow examination of many cases. The simplified models provide only approximate results.

In this paper, we describe an analysis technique for studying the performance of checkpointing schemes with task duplication. The technique, which is based on modeling the scheme as a discrete time *Markov Reward Model* (MRM) [8], provides a means to evaluate important parameters in the performance of a scheme. It provides a way to compare various schemes and select optimal values for some parameters of the scheme, like the number of checkpoints [19].

The proposed analysis technique is used to compare four checkpointing schemes: TMR-F [11], DMR-B-2 [11], DMR-F-1 [11], and RFCS [15]. We evaluate two quantities: the average execution time of a task and the processor work done to complete a task. The execution time of a task is defined as the total elapsed time from the beginning of the execution of the task until the last checkpoint is compared correctly. This parameter is important in real-time systems, where fast response is desired. We show that the number of processors used to implement the scheme has a major effect on the average execution time, while the complexity of the scheme has only a minor effect. Out of the four schemes examined in this paper, the TMR-F scheme, which uses three processors and a simple fault-masking recovery technique, is the quickest. The DMR-F-1 and RFCS schemes, which use two processors during normal execution and add spare processors during fault recovery, are slower than TMR-F but quicker than the DMR-B-2 scheme, which always uses two processors.

The processor work to complete a task depends not only on the time to execute the task but also on the number of processors used. It is defined as the sum of the time each of the processors is used by the scheme. This parameter is important in transactions systems, where high availability is important. In these types of systems, reducing the processor work to complete a task means increasing the total throughput of the system. We show that schemes with low execution time are not work efficient, and that the lowest work is done using schemes that use a small number of processors, and have higher execution time. The processor work results of the four schemes examined here were the reverse of the execution time results. The DMR-B-2 scheme has the lowest processor work, while the TMR-F scheme has the highest processor work.

There are some cases where a big difference in the time it takes to perform various operations can cause the schemes to behave differently than described above. Those cases can still be analyzed with the technique described in this paper. For example, when workstations connected by a LAN are used to implement the schemes, operations that involve more than one workstation, and need the LAN, take longer time to execute than operations that can be done locally. In this case, the DMR-B-2 scheme that uses the LAN only lightly is quicker than the TMR-F scheme.

The rest of the paper is organized as follows. Section 2 describes the analysis technique, using Double Modular Redundant

- *A. Ziv is with the Haifa Research Laboratory, IBM Israel, Science and Technology, MATAM, Haifa 31905, Israel. E-mail: aziv@vnet.ibm.com.*
- *J. Bruck is with the California Institute of Technology, Mail Code 136-93, Pasadena, CA 91125. E-mail: bruck@paradise.caltech.edu.*

For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 106033.

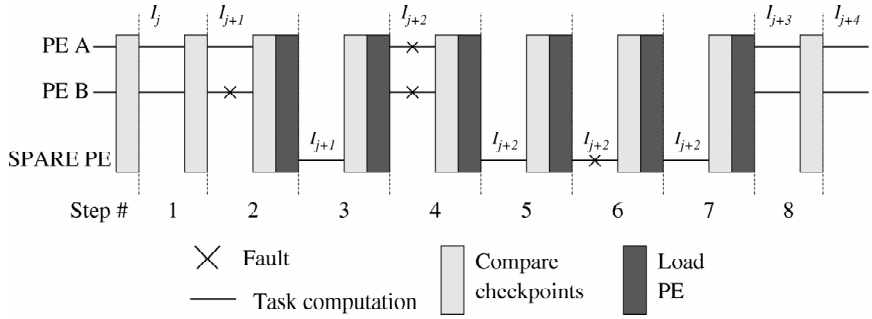

Fig. 1. Example of execution with the DMR-B-1 scheme.

scheme with backward recovery and a single recovery processor (DMR-B-1) [11] as an example. In Section 3, we compare the average execution time and the processor work of four checkpointing schemes. Section 4 concludes the paper.

2 Analysis Technique

The analysis of the schemes is based on the analysis of a discrete time *Markov Reward Model* (MRM) [8]. In the Markov Reward Model used in this paper, each transition edge of the Markov chain has a reward level associated with it. The properties of the rewards of the Markov chain are used to evaluate the measures of interest. Markov Reward Models are often used in evaluating the performance of computing systems. Smith and Trivedi [16] give examples of the use of MRM in evaluating reliability and performance of parallel computer, task completion time in faulty systems, and properties of queuing systems. Others, like [2], [5], [17], use MRM to evaluate various aspects of computer system performance.

The analysis of the schemes is done in four steps: Building the *extended* state-machine of the scheme, assigning rewards to the transition edges of the state machine, assigning transition probabilities to the transition edges according to the fault models, and solving the Markov chain created by the first three steps to get the desired analysis. Next, we describe the four steps in more detail, using, as an example, the DMR-B-1 scheme [11].

In the DMR-B-1 scheme, the task is executed by two processors in parallel. At the end of each interval, the states of both processors (or signatures of them) are compared. If they match, then a correct execution is assumed, and the execution of the next interval starts. In case the states do not match, a new processor executes the interval and its state is compared to all the states of the previous executions of the interval until two identical states are found.

Fig. 1 gives an example of execution of a task with the DMR-B-1 scheme. In the figure, the horizontal lines represent execution of the task code by the processors assigned to it (PE *A* and *B*) or the spare processor, with the interval number indicated by the $I_{...}$ above the horizontal lines. The boxes represent operations done by the system to achieve fault tolerance. In Step 2 of the execution, while executing interval I_{j+1}, a fault occurred in processor *B*, so, in Step 3, a spare processor repeats the execution of the same interval. After it finishes, its state matches the state of processor *A*, hence the interval is verified, and normal execution can be resumed. During Step 4, both processors have faults and the spare processor has to produce two correct executions before fault recovery is achieved and normal execution can be resumed. As the spare processor has a fault during the recovery process (in Step 6), it takes three steps to complete the recovery.

2.1 Building the State-Machine

The first step in analyzing a fault recovery scheme is to build the state-machine that describes the operation of the scheme. Since we want to model the scheme as an MRM, the state machine that describes the operation of the scheme needs to have the Markov property, namely, the transition from state *i* to state *j* cannot depend on how or when state *i* was reached. The problem is that, from the system point of view, the behavior of the scheme is not necessarily Markovian. For example, in the DMR-B-1 scheme, the transition from the recovery mode to normal operation is done when two correct execution of the same interval were completed. Since the system does not know how many correct executions occurred so far, the transition from the recovery mode to normal operation depends not only on correct execution at the current step, but also on a correct execution some time in the past.

To overcome this problem, we build an *extended* state-machine. The extended state-machine describes the behavior of the scheme in the eyes of an external viewer, who can observe the faults that occurred during a step. Two fault patterns that are not distinguishable in the scheme, but might later cause different actions, cause transitions to different states in the extended state-machine. For example, when two processors execute the same interval, and their states do not match at the end, the scheme can not tell if the fault occurred in one of the processors or in both. The number of faults might affect the ability of the scheme to recover from the faults, and thus should cause transitions to different states in the extended state-machine.

Each transition in the state-machine represents one step, and a transition is done at the end of each step. Because of the way the state-machine is constructed, the transition is determined only by the current state and the faults that occur during the current step and, therefore, it holds the Markov property.

Note that the first step in the analysis depends only on the scheme and is totally independent of the implementation details of the scheme or the fault model.

In the DMR-B-1 scheme, the operation of the scheme has two basic modes. The first mode is the normal operation mode, where two processors are executing the task in parallel. The second mode is the fault recovery mode, where a single processor tries to find a match to an unverified checkpoint.

Fig. 2 shows the extended state-machine for the DMR-B-1 scheme. The state-machine has two different fault recovery states; the first state has no correct execution of the current interval so far, and the second state has a single correct execution. State 2 in the machine is the normal execution state, and States 0 and 1 are the fault recovery states with the respective number of correct executions. Table 1 describes all the possible transitions in the state-machine with their properties. The first two columns in the table describe, for each possible transition, the event that causes it. The rest of the columns are explained later in the section.

The execution of the scheme starts at State 2 and, if no faults occur, it remains there, or, in other words, a transition is made via edge 0. If a mismatch between the states of the processors is found, a transition to a fault recovery state is made. As the external observer knows how many faults occurred, it knows if it has to move along edge 2 to State 0 (faults in both processors), or along edge 1 to State 1 (one fault only).

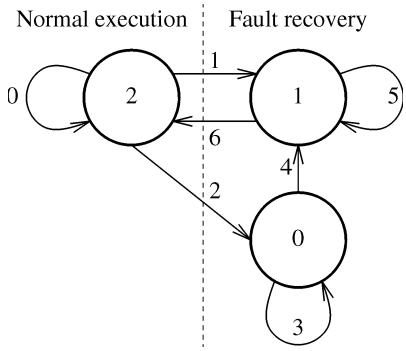

Fig. 2. Extended state-machine for the DMR-B-1 scheme.

TABLE 1
TRANSITION DESCRIPTION
FOR THE DMR-B-1 EXTENDED STATE-MACHINE

Edge	Event	Interval completion (v_i)	Time to execute (t_i)	Transition Prob. (p_i)
0	No faults	1	$t_I + t_{ck}$	$(1-F)^2$
1	One fault	0	$t_I + t_{ck} + t_{ld}$	$2F(1-F)$
2	Two faults	0	$t_I + t_{ck} + t_{ld}$	F^2
3	Fault	0	$t_I + t_{ck} + t_{ld}$	F
4	No fault	0	$t_I + t_{ck} + t_{ld}$	$(1-F)$
5	Fault	0	$t_I + t_{ck} + t_{ld}$	F
6	No fault	1	$t_I + t_{ck} + t_{ld}$	$(1-F)$

In the fault recovery states, the recovery processor executes the task again, and, every time it fails, it remains in the same state (transition via edge 3 or 5). When a correct execution is completed, a transition to the next state is made.

For example, the execution of Fig. 1 causes the following transitions (the number above the arrows are the edges that are used for the transitions)

$$2 \xrightarrow{0} 2 \xrightarrow{1} 1 \xrightarrow{6} 2 \xrightarrow{2} 0 \xrightarrow{4} 1 \xrightarrow{5} 1 \xrightarrow{6} 2 \xrightarrow{0} 2.$$

2.2 Assigning Rewards to the Transition Edges

After the state-machine that describes the operation of the scheme is built, each transition is associated with a set of properties, called rewards. The type of rewards that are used depends on the measures of interest, and the values of the rewards depend on the event that caused the transition. The rewards are used to evaluate the measures of interest related to the scheme. In this paper, we are interested in the execution time of the schemes. Two rewards are used for that. Other measures, such as the number of checkpoints stored in the stable storage, can also be viewed as rewards and analyzed using the technique described here. The two quantities we use for execution time analysis are:

1) v_i—The amount of useful work that is done during the transition. We measure the useful work as the number of intervals whose checkpoints were matched as a result of the event that caused the transition.
2) t_i—The time it takes to complete the step that corresponds to the transition. A step starts when the processor(s) start to execute an interval, and ends the next time an interval is ready to be executed. The time it takes to complete a step is the time to perform all the operations of that step. Each step includes at least the execution of the interval, denoted as t_I, and the comparison of the states at the end of the interval, denoted as t_{ck}. Some steps may include other operations like loading a spare processor, copying the state from one processor to another, etc.

The third, and fourth columns in Table 1 show the values of the rewards of interest, v_i and t_i, for the DMR-B-1 scheme.

In DMR-B-1, there are two transitions that complete the execution of an interval, and hence do useful work. The first transition is edge 0, where no fault occurred during normal execution. The second one is the transition out of the recovery mode, edge 6. The value of v_i for those two edges is one. All other transitions do not do any useful work and, thus, their value of v_i is zero.

The time to complete any step in the DMR-B-1 scheme includes the time to execute the interval and compare the checkpoints at the end. We assume that a spare processor is loaded before every step in the fault recovery mode, and the main processors are loaded when the recovery is completed. Hence, all the edges have execution time of $t_I + t_{ck} + t_{ld}$, except edge 0 that has execution time of $t_0 = t_I + t_{ck}$.

2.3 Creating the Markov Chain

The third step in the analysis is assigning transition probabilities to each of the transitions in the state-machine constructed in the first step. Each edge i is assigned a probability p_i, which is the probability of the event that causes the transition via that edge occurring.

The probabilities assigned to the edges are determined by the fault model. In the simplest case, it is assumed that the fault patterns do not change with time and, thus, the transition probabilities are constants. More complex models assume that the fault pattern changes with time, or is a random process. In this case, the probabilities of transitions are functions of time or random processes.

The transition probabilities out of a state do not depend on the way this state was reached. Hence, the state-machine with the transition probabilities corresponds to a Markov chain. Together with the properties of the transitions, or the rewards, described earlier, a Markov Reward Model is created. The analysis of this MRM provides results related to the fault recovery scheme.

In the example here, we assume that the fault pattern does not change with time and, thus, the transition probabilities are constants. We also assume that the faults in different processors are independent of each other. This fault model is used in [11] and [15]. In this model, F is the probability that a processor will have a fault while executing an interval. The probabilities of the transitions using this fault model appear in the last column of Table 1.

2.4 Analyzing the Scheme Using the MRM

After constructing the MRM induced by the fault recovery scheme and the fault model, its analysis provides the required results. There are two ways to analyze a Markov chain, transient analysis and steady-state or limiting analysis. In steady-state analysis, we look at the state probabilities in the limit as $t \to \infty$. Therefore, it is less accurate for finite length tasks, but it is simpler. Comparison between results obtained using transient analysis and steady-state analysis show negligible difference between them for a large range of fault rates. Therefore, in this paper, we use the steady-state analysis. A detailed discussion on analysis of Markov chains can be found in many text books, such as [9].

Steady-state analysis provides us with $\vec{e}$, the vector of steady-state probabilities of transitions, and the average reward R for edge reward vector $\vec{r}$, given by $\vec{e} \cdot \vec{r}$. These values can be used to perform time analysis of a checkpointing scheme of a task with n intervals. We next show how to calculate the average execution time and processors work of a task.

2.4.1 Average Execution Time

In steady-state, T_1, the average time to complete a single interval is equal to the average time to complete a single step, given by the reward T, divided by the average progress in intervals in a single step, i.e., the reward V. $\overline{T}$, the average execution time of a task with n intervals, is n times this quantity. Therefore, the average times to complete one interval and the whole task of n intervals are:

$$T_1 = \frac{\sum_i t_i e_i}{\sum_i v_i e_i} \text{ and } \overline{T} = n \cdot T_1 = n \frac{\sum_i t_i e_i}{\sum_i v_i e_i}.$$

2.4.2 Average Processor Work

The processor work to complete a task considers not only the execution time of each step, but also the number of processors used. To calculate the average processor work, we replace the step execution time reward vector $\vec{t}$ with $\vec{w}$, a reward vector for the work done in each transition. The values of $\vec{w}$ are given by the execution time reward values times the number of processors used in each transition. For example, in DMR-B-1 the vector of number of processors used in the transitions is {2, 2, 2, 1, 1, 1, 1}, and the processor work vector $\vec{w}$ equals to {2, 2, 2, 1, 1, 1, 1}$\vec{t}$. The average work to complete a task with n intervals is given by

$$\overline{W} = n \cdot \frac{\sum_i w_i e_i}{\sum_i v_i e_i}.$$

For the DMR-B-1 scheme, the average execution time and processor work for a task with n intervals are given by

$$\overline{T} = \frac{(1+F)(nt_I + nt_{ck}) + (4F - 3F^2 + F^3)nt_{ld}}{1-F},$$

$$\overline{W} = \frac{2n(t_I + t_{ck} + (3F - 3F^2 + F^3)t_{ld})}{1-F}.$$

3 Scheme Comparison

In this section, the analysis technique is used to compare between four existing checkpointing schemes. The schemes we compare are Triple Modular Redundant with checkpointing (TMR-F) [11], Double Modular Redundant with look-back recovery and two recovery processors (DMR-B-2) [11], Double Modular Redundant with forward recovery and one recovery processor (DMR-F-1) [11], and Roll-Forward Checkpointing Scheme (RFCS) [15]. A short description of the schemes is given here. A more detailed description and the analysis of those schemes can be found in [19].

The simplest scheme is the TMR-F scheme [11]. In this scheme, the task is executed by three processors, all of them executing the same interval. A fault in a single processor can be recovered without a rollback because two processors with correct execution still agree on the checkpoint. If faults occur in more than one processor, all the processors are rolled back and execute the same interval again.

The DMR-B-2 scheme is described by Long et al. in [11]. In this scheme, two processors execute the task. Whenever a fault occurs, both processors are rolled back and execute the same interval again. The difference between this scheme and simple rollback schemes, like TMR-F, is that all the unverified checkpoints are stored and compared, not just the checkpoints of the last step. Hence, two steps with a single fault are enough to verify an interval.

The next two schemes, DMR-F-1 and RFCS, use spare processors and the roll-forward recovery technique in order to avoid a rollback [14]. In the DMR-F-1 scheme, suggested by Long et al. in [11], two processors are used during fault-free steps. Three additional spare processors are added for a single step after each fault to try to recover without a rollback. The states of the two processors that are currently executing the task are copied to two of the spare processors. The third spare processor is loaded with the last verified checkpoint and tries to verify the faulty checkpoint. If it fails, either because it had a fault or because both processors had faults in the previous step, a rollback is done. If the verification succeeds, then no rollback is done and the processor with the correct checkpoint and the one that this checkpoint was copied to continue to execute the task.

Pradhan and Vaidya [15] describe another roll-forward scheme called Roll-Forward Checkpointing Scheme (RFCS). In this scheme, as in DMR-F-1, a spare processor is used in fault recovery in order to avoid rollback. The difference between the schemes is that RFCS uses only one spare processor and the recovery takes two steps instead of one step in DMR-F-1. In the first step of fault recovery, the spare processor is loaded with the last verified checkpoint and it tries to verify the current checkpoint, while the two regular processors continue with the normal execution. If the spare processor succeeds in verifying the first checkpoint, the state of the correct processor is copied to the faulty processor. In the next step, the spare processor tries to verify the next checkpoint that has only one correct execution.

We compare here two properties of the schemes. The first property is the average execution time of a task using the scheme. The second property is the average processor work used to complete the execution of a task using the scheme. We assume that faults occur according to a Poisson random process with rate λ, i.e., the probability of a fault in a processor during the execution of an interval is $F = 1 - e^{-\lambda \cdot t_I}$. We also assume that the number of checkpoints in the task is chosen such that the best possible result is achieved, given the scheme and the fault rate λ.

The average execution time of a task is important in real-time systems where fast response is desired. We show here that the average execution time is affected mostly by the number of processors used by the scheme, and the complexity of the scheme has only a minor effect.

The processor work to complete the execution of a task is the sum, over all processors used by the scheme to complete the task, of the time they were in use. The processor work is important in transaction systems, where high availability of the system is required and, thus, the system should use as few resources as possible. We show here that the best processor work is achieved when a small number of processors are used and, again, the complexity of the scheme has only a minor effect.

3.1 Simplified Model

The behavior of the schemes is greatly affected by their exact implementation and the architecture of the parallel computer. These parameters affect the time it takes to execute the operations that are needed at the end of each step, like comparing checkpoints and rolling back. To obtain general properties of the schemes without the influence of a specific implementation, we use a simpler model than the one used in Section 2. In the simplified model, the time to execute each step is $t_I + t_{oh}$, where t_{oh} is the overhead time required by the scheme. This overhead time is the same for all the transitions of the state-machine of the scheme. It is also assumed to be the same for all schemes.

Using the analysis technique described in Section 2, we calculated the average execution time of the four schemes considered in this section for a task of length one with n checkpoints [19]:

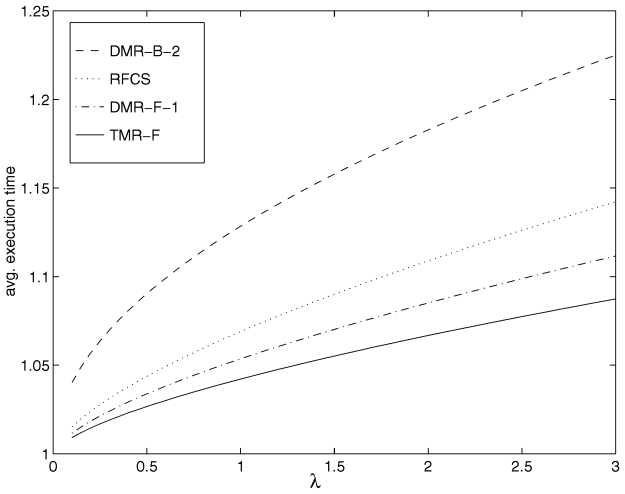

Fig. 3. Average execution time with optimal checkpoints.

$$\overline{T}_{\text{TMR-F}} = \frac{1}{1-\left(3F^2-2F^3\right)}\cdot\left(1+nt_{oh}\right),$$

$$\overline{T}_{\text{DMR-B-2}} = \frac{1+3F}{(1-F)(1+F)^2}\cdot\left(1+nt_{oh}\right),$$

$$\overline{T}_{\text{DMR-F-1}} = \frac{1+3F^2-2F^3}{1-3F^2+2F^3}\cdot\left(1+nt_{oh}\right),$$

$$\overline{T}_{\text{RFCS}} = \frac{1+2F+3F^2-10F^3+8F^4-2F^5}{1+2F-11F^2+14F^3-8F^4+2F^5}\cdot\left(1+nt_{oh}\right).$$

Fig. 3 shows the average execution time of a task using each of the four schemes, with overhead time of $t_{oh} = 0.002$ for each step. The number of checkpoints for each scheme is chosen such that its average execution time is minimized [19].

The figure shows that the TMR-F scheme, despite being the simplest of the four schemes, has the lowest execution time. The TMR-F scheme has better execution time because it is using more processors than the other schemes and, thus, has a much lower probability of failing to find two matching checkpoints. The DMR-B-2 scheme is the worst because it uses only two processors, and does not use spare processors to try to overcome failures. The RFCS and DMR-F-1 schemes use spare processors during fault recovery and, thus, have better performance than DMR-B-2.

The second property we compared is the average processor work. Applying the analysis technique to the four schemes gives the following average processor work for task of length one with n checkpoints:

$$\overline{W}_{\text{TMR-f}} = \frac{3}{1-\left(3F^2-2F^3\right)}\cdot\left(1+nt_{oh}\right),$$

$$\overline{W}_{\text{DMR-B-2}} = \frac{2+6F}{(1-F)(1+F)^2}\cdot\left(1+nt_{oh}\right),$$

$$\overline{W}_{\text{DMR-F-1}} = \frac{2+6F+3F^2-4F^3}{1-3F^2+2F^3}\cdot\left(1+nt_{oh}\right),$$

$$\overline{W}_{\text{RFCS}} = \frac{2+8F+F^2-18F^3+16F^4-4F^5}{1+2F-11F^2+14F^3-8F^4+2F^5}\cdot\left(1+nt_{oh}\right).$$

The average processor work of a task of length one with overhead time of $t_{oh} = 0.002$ for the four schemes is shown in Fig. 4.

The results here are the reverse of the results in the average execution time. The best scheme here is the DMR-B-2, which always uses only two processors. The RFCS and DMR-F-1, which use two processors during normal execution and add spare processors during fault recovery, require more processor work. The

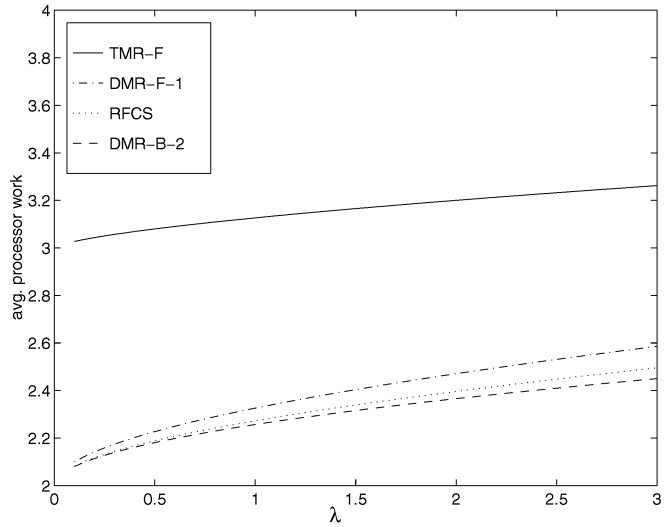

Fig. 4. Average processor work with optimal checkpoints.

TMR-F scheme, which uses three processors, is the worst scheme.

3.2 Precise Model

When a more precise model is used, in which the time to perform each operation is used (as in the analysis done in Section 2), the results shown here for the simplified model are still valid for a large range of scheme parameters. However, there are cases where a big difference in scheme parameters can cause different behaviors of the schemes than those described for the simplified model. These cases can still be analyzed with the technique described in this paper, by using the more precise model described in Section 2, instead of the equation used in the simplified model.

For example, consider the following case: Workstations connected by a LAN are used to implement the schemes. Each workstation saves its own checkpoint states, and sends only a short signature of them to the other workstations for comparison. In this implementation, operations that are done within a workstation can be completed relatively quickly, while operations that involve more than one workstation, and need the LAN, take much longer to execute. In this case, schemes that do not use the network heavily have lower execution time than those which do. Specifically, the slowest scheme under the general model, the DMR-B-2 scheme, which uses the network only for state comparison, can become the quickest scheme under these conditions. Fig. 5 shows the execution time of a task when $t_{ck} = 0.001$, the time to roll back a processor to a state previously saved on its local disk is 0.001, and the time to copy a state from one processor to another is 0.03. The execution time is shown for the case when the optimal number of checkpoints is used. In the case described by the example, schemes that rely mainly on the local storage perform better than schemes that need the network to copy states between processors. Specifically, the DMR-B-2 scheme, which uses mostly the local disk, is the quickest scheme after the failure rate, λ, reaches some critical value that requires the other schemes to use the network heavily.

Since the system on which the scheme is implemented and the exact details of the implementation can have a major effect on the performance of the schemes, as the LAN example above shows, the precise model with the exact time to perform each operation should be used when a specific system is considered.

4 Conclusions

In this paper, we have proposed a technique to analyze the performance of checkpointing schemes. The proposed technique is based on modeling the schemes under a given fault model as a

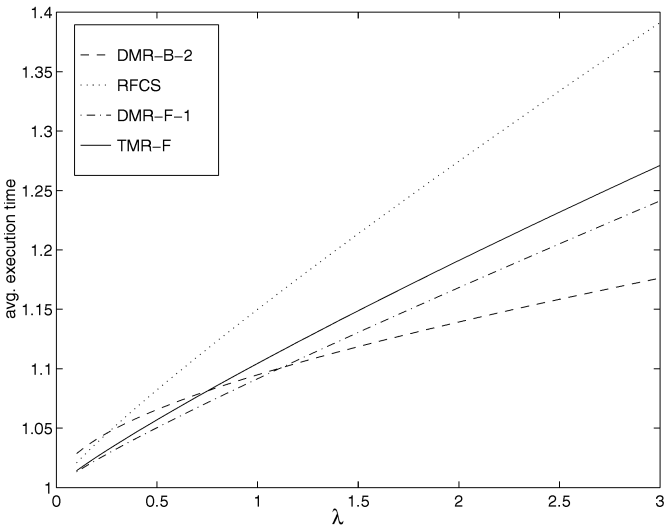

Fig. 5. Average execution time for the workstations example.

Markov Reward Model, and evaluating the required measures by analyzing the MRM.

We compared the average execution time of a task and the total processor work done for four known checkpointing schemes. The comparison shows that, generally, the number of processors has a major effect on both quantities. When a scheme uses more processors, its execution time decreases, while the total work increases. The complexity of the scheme has only a minor effect on its performance. In some cases, when there is a big difference between the time it takes to perform different operations, the general comparison results are no longer true. However, the proposed technique can still handle these cases and give correct results for them.

The proposed technique is not limited to the schemes described in this paper, or to the fault model used here. It can be used to analyze any checkpointing scheme with task duplication, with various fault models. The proposed technique can be also used to provide analytical answers to problems that haven't been dealt with before or were handled by a simulation study. An example of such problems is deriving the number of checkpoints that minimizes the average execution time.

ACKNOWLEDGMENTS

The research reported in this paper was supported in part by the U.S. National Science Foundation Young Investigator Award CCR-9457811, by the Sloan Research Fellowship, and by DARPA and BMDO through an agreement with NASA/OSAT.

REFERENCES

[1] P. Agrawal, "Fault Tolerance in Multiprocessor Systems without Dedicated Redundancy," *IEEE Trans. Computers*, vol. 37, no. 3, pp. 358-362, Mar. 1988.
[2] A. Bobbio, "A Multi-Reward Stochastic Model for the Completion Time of Parallel Tasks," *Proc. 13th Int'l Teletraffic Congress*, pp. 577-582, 1991.
[3] K.M. Chandy and C.V. Ramamoorthy, "Rollback and Recovery Strategies for Computer Programs," *IEEE Trans. Computers*, vol. 21, no. 6, pp. 546-556, June 1972.
[4] E.G. Coffman and E.N. Gilbert, "Optimal Strategies for Scheduling Checkpoints and Preventive Maintenance," *IEEE Trans. Reliability*, vol. 39, pp. 9-18, Apr. 1990.
[5] L. Donatiello and V. Grassi, "On Evaluating the Cumulative Performance Distribution of Fault-Tolerant Computer Systems," *IEEE Trans. Computers*, vol. 40, no. 11, pp. 1,301-1,307, Nov. 1991.
[6] A. Duda, "The Effects of Checkpointing on Program Execution Time," *Information Processing Letters*, vol. 16, pp. 221-229, June 1983.
[7] E. Gelenbe, "On the Optimum Checkpoint Interval," *J. ACM*, vol. 26, pp. 259-270, Apr. 1979.
[8] R.A. Howard, *Dynamic Probabilistic Systems Vol II: Semi Markov and Decision Processes*. John Wiley, 1971.
[9] L. Kleinrock. *Queueing Systems, Vol. I: Theory*. John Wiley, 1975.
[10] V.G. Kulkarni, V.F. Nicola, and K.S. Trivedi, "Effects of Checkpointing and Queueing on Program Performance," *Comm. Statistics—Stochastic Models*, vol. 6, pp. 615-648, Apr. 1990.
[11] J. Long, W.K. Fuchs, and J.A. Abraham, "Forward Recovery Using Checkpointing in Parallel Systems," *Proc. 19th Int'l Conf. Parallel Processing*, pp. 272-275, Aug. 1990.
[12] J. Long, W.K. Fuchs, and J.A. Abraham, "Compiler-Assisted Static Checkpoint Insertion," *Proc. 22nd IEEE Int'l Symp. Fault-Tolerant Computing*, pp. 58-65, July 1992.
[13] J. Long, W.K. Fuchs, and J.A. Abraham, "Implementing Forward Recovery Using Checkpoints in Distributed Systems," *Dependable Computing for Critical Applications 2*, R.D. Schlichting and J.F. Meyer, eds., pp. 27-46. Springer-Verlag, 1992.
[14] D.K. Pradhan, "Redundancy Schemes for Recovery," Technical Report TR-89-cse-16, Electrical and Computer Eng. Dept., Univ. of Massachusetts, Amherst, 1989.
[15] D.K. Pradhan and N.H. Vaidya, "Roll-Forward Checkpointing Scheme: Concurrent Retry with Nondedicated Spares," *Proc. IEEE Workshop Fault-Tolerant Parallel and Distributed Systems*, pp. 166-174, July 1992.
[16] R.M. Smith and K.S. Trivedi, "The Analysis of Computer Systems Using Markov Reward Processes," *Stochastic Analysis of Computer and Communication Systems*, H. Takagi, ed., pp. 589–629. North-Holland, 1990.
[17] D. Tang and R.K. Iyer, "Dependability Measurement and Modeling of a Multicomputer System," *IEEE Trans. Computers*, vol. 42, no. 1, pp. 62-75, Jan. 1993.
[18] S. Toueg and Ö. Babaoglu, "On the Optimum Checkpoint Selection Problem," *SIAM J. Computing*, vol. 13, pp. 630-649, Aug. 1984.
[19] A. Ziv, "Analysis and Performance Optimization of Checkpointing Schemes with Task Duplication," PhD thesis, Stanford Univ., 1995.